CS3025 Compiladores

Laboratorio 8.2 – 3 octubre 2023

Enviroments y declaraciones II

El folder lab8.2 contiene la solución del laboratorio 8.1.
Estos programas implementan el parser, printer e interpreprete de un lenguaje imperativo con declaraciones. El lenguaje tiene la siguiente sintaxis:

```
Program ::= VarDecList StatementList
VarDecList ::= (VarDec)*
VarDec ::= "var" Type VarList ";"
Type ::= id
VarList :: id ("," id)*
StatementList ::= Stm (";" Stm)* …
```

La sintaxis del lenguaje hace posible agregar declaraciones antes de la lista de sentencias de nuestro programa. Por ejemplo, podemos escribir:

```
int x, y;
bool a;
A b;
x = 2;
y = x+4;
z = x > y
```

Notese que todavía no ponemos restricción en el nombre del tipo (int, bool) – por ahora puede ser cualquier identificador. El ejemplo de arriba es un programa correcto de acuerdo a la gramática pero retorna error al tratar de ser ejecutado (semantica). Es necesario agregar la declaración de z:

```
int x, y;
bool z;
x = 2;
y = x+4;
z = x > y
```

Compilar y probar el programa con los ejemplos `ejemplo11.imp`, `ejemplo22.imp` y `ejemplo33.cpp`.

1.0 Agregando declaraciones en otros puntos del programa.

Nuestro lenguaje solo trabaja con 1 nivel de variables – todas las variables son globales (global scope). El siguiente paso es hacer posible la declaración de variables al comienzo de sentencias compuestas como while-do e if-then-else. Para esto definiremos una clase sintáctica nueva, `Body`, que estará compuesta de una lista de declaraciones más una lista de sentencias. La nueva gramática del lenguaje es:

```
Program ::= Body
Body ::= VarDecList StatementList
VarDecList ::= (VarDec)*
VarDec ::= “var” Type VarList “;”
Type ::= id
VarList :: id (“,” id)*
StatementList ::= Stm (“;” Stm)* …
```

La modificada gramática de sentencias es:

```
Stm ::=   id “=” CExp |
          “print” “(“ CExp “)” |
          “if” CExp “then” Body [“else” Body] “endif” |
          while” CExp “do” Body “endwhile”
```

Modificar las clases del AST, parser, visitor, interprete y printer para implementar los cambios a la gramática.

Se sugieren los siguientes primeros pasos:

- Crear una nueva clase `Body`, con componentes `VarDecList` y `StatementList`.
- Cambiar los campos de `Program`: solo necesita una instancia de Body.
- En las clases `IfStatement` y `WhileStatement`, cambiar los `StatementList` and `Body`.
- Modificar/agregar constructores, destructores y metodos `accept`.
- Agregar el método `visit` para `Body` en `ImpVisitor`.

En este punto, es posible compilar imp.cpp (-c) para verificar que todos está bien.

En el Parser:

- Agregar la declaración de `Body* parseBody()`.
- Agregar la implementación de `parseBody`() – hace lo mismo que hacia `parseProgram`() – y modificar `parseProgram` y `parseStatement` (para los casos `IfStatement` y `WhileStatement`): cambiar los `parseStatementList` y `StatementList` por `parseBody` y `Body`.

Luego continuar con los siguientes pasos:

- En `imp_printer.hh`, agregar el método `visit` para `Body`.

- En imp_printer.cpp, agregar `ImpPrinter::visit(Body * b)` y modificar los métodos visit (en la mayoria de los casos, actualizar los nombres de las variables) para `Program`, `IfStatement` y `WhileStatement`.
- Probar `ImpPrinter`: en tes_imp_dec.cpp, comentar todas las referencias a `ImpPrinter` (include también). Compilar y correr los ejemplos.

Completar imp_interpreter.hh / imp_interpreter.cpp.

2.0 Agregar Funciones

La definición de `Body` nos permite extender con facilidad la gramática a funciones. Por ejemplo, podríamos declarar funciones de la siguiente manera:

```
FunDec ::= "fun" id "(" ParamDecList ")" ":" Type Body "endfun"
ParamDecList ::= ParamDec ("," ParamDec)* | epsilon
ParamDec ::= Type id
```

Y definir un programa como:

```
Program ::= VarDecList FunDecList
```

Donde una de las funciones tiene que ser `main`, por ejemplo.

La implementación de este lenguaje será nuestro proximo paso.